



User Interface Programming with Python

Peter Hartford

Table of Contents



- When should you create a User Interface?
- Part 1: File Structure/Preliminary GUI
- Part 2: Our First Function
- Part 3: A Long Running Function
- Part 4: Implement the POD

This should teach all of the basics

Fact of the day: some people say “Gee u eye”,
Peter says “Gooey” because it sounds funnier

When should you create a user interface?



➤ User interfaces are a somewhat large undertaking

- ☐ Can easily become a waste of time
- ☐ Sometimes better to just have a script



Instead of asking whether you can (you can), ask whether you should...



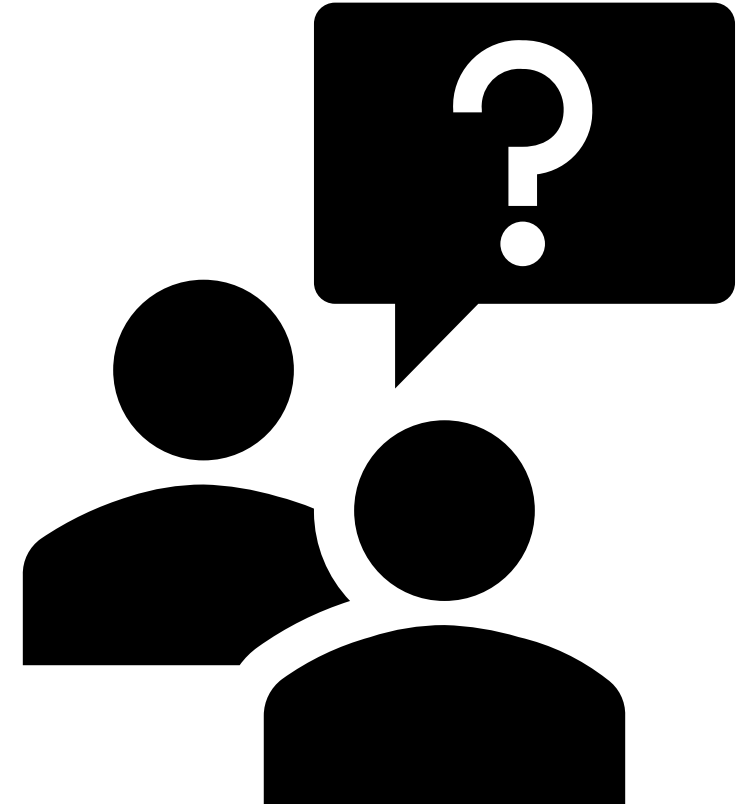
When should you create a user interface?

➤ User interfaces are a somewhat large undertaking

- ☐ Can easily become a waste of time
- ☐ Sometimes better to just have a script

➤ Ask yourself

- ☐ Am I using this script often (e.g. recording data)? ✓
- ☐ Am I repeatedly changing parameters and adjusting things? ✓
- ☐ Am I continuously changing the functions? ✗
- ☐ Is someone going to use your code? ✓
- ☐ Is someone going to **edit** your code? ✗



➤ Post processing scripts usually don't need a GUI

Instead of asking whether you can (you can), ask whether you should...



Part 1: File Structure/Preliminary GUI

Example Script: POD Filter for PIV Images



➤ Why is this a good example?

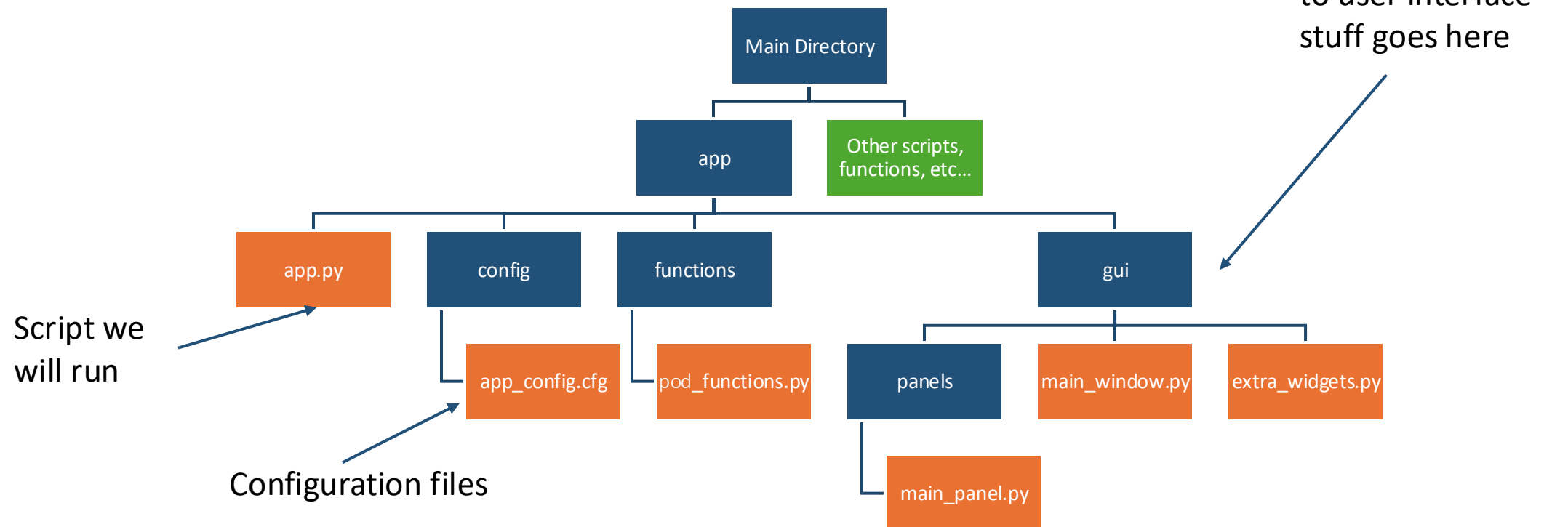
- ❑ Has a few parameters that need to be changed
- ❑ Can save and load images
- ❑ The script takes time to run (more on this later)...
- ❑ Ability to extend to interactive plotting

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Sep 26 11:37:58 2019
4
5 @author: mendez, torres
6 """
7
8 import os # This is to understand which separator in the paths (/ or \)
9
10 import matplotlib.image as mpimg
11 import numpy as np # This is for doing math
12 from skimage.io import imread, imshow, imsave # this is for Matlab users
13
14 def generate_filename(folder, number, pair, pic_format):
15     """
16     simple function to generate filenames
17     :param folder: where to look or save
18     :param number: number of the picture
19     :param pair: a or b
20     :param pic_format: format of the image
21     :return:
22     """
23     return folder + os.sep + 'A' + '%03d' % (number + 1) + pair + '.' + pic_format
24
25
26 # Image Cropping, Flipping and pre-processing using the 1POD mode removal
27 # For more advanced version, see https://seis.bristol.ac.uk/~aexrt/PIVPODPreprocessing/
28 # To do list: Implement Frequency based filter.
29
30 # Ensure current working directory
31 cwd = os.path.dirname(os.path.realpath(__file__))
32 os.chdir(cwd) #chdir used for change directory
33 print(os.getcwd())
34
35 # Folder in
36 FOL_IN = 'RAW_IMAGES'
37 # Processing Images
38 FOL_OUT = 'Pre_Pro_PIV_IMAGES' # Where will the result be
39 if not os.path.exists(FOL_OUT):
40     os.mkdir(FOL_OUT)
41
42
43 # To define the crop area you will usually need to load at least one image
44 # and plot on it a rectangle
45 Num = 10 # This is the number of the image
46 Name=generate_filename(FOL_IN, 10, 'a', 'tif')
47
48 Im = imread(Name, as_gray=True)
49 ny, nx = Im.shape
```

Folder/File Structure



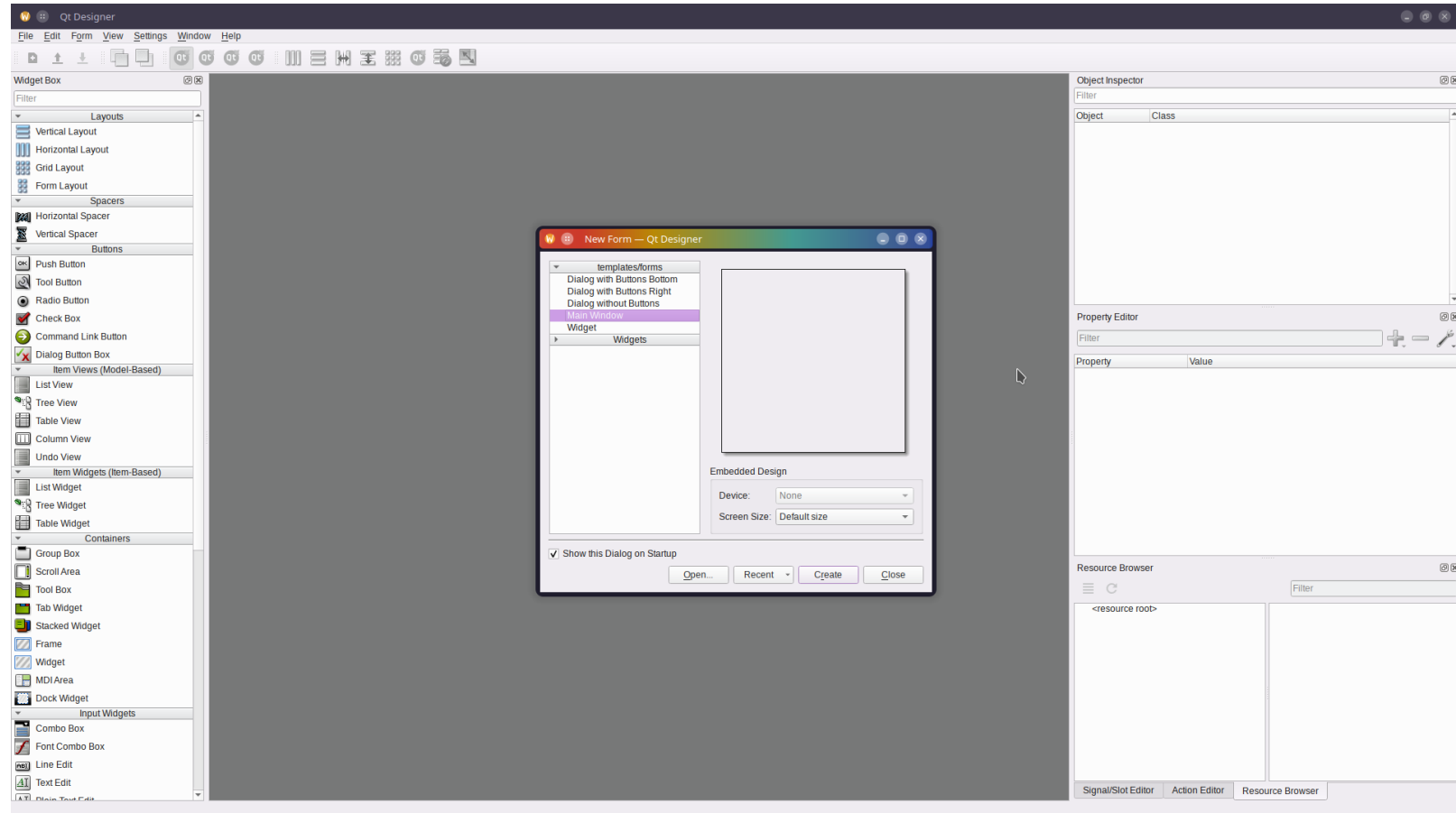
- File organization is key since we are creating multiple python files
- Usually use something like this
 - ❑ The files we will create are in orange, folders in blue
 - ❑ This is already started for you



Getting Started: Qt Designer



- First thing we will generate is a panel file
- Open Qt Designer
 - ❑ In the terminal type “pyqt5-tools designer”
 - ❑ Qt Designer should open with a new form dialog
 - ❑ Create a "main window" form



Navigating QT Designer



The screenshot shows the Qt Designer application window. The interface is divided into several panels:

- Widget Box (Left):** Contains a list of widget types categorized into Layouts, Spacers, Buttons, Item Views (Model-Based), Item Widgets (Item-Based), and Containers. A red box highlights this entire section with the text: "All of the widget types are here".
- Main Window (Center):** A large canvas for designing the user interface. A red box highlights this area with the text: "Our window layout is here".
- Object Inspector (Top Right):** Displays a tree view of the widget hierarchy. A red box highlights this area with the text: "Widget names and trees are here".
- Property Editor (Bottom Right):** Displays the properties of the selected widget. A red box highlights this area with the text: "Widget properties are here".
- Resource Browser (Bottom):** Displays the resource files used in the application.

The Object Inspector shows the following hierarchy:

Object	Class
MainWindow	QMainWindow
centralwidget	QWidget
menubar	QMenuBar
statusbar	QStatusBar

The Property Editor shows the following properties for the selected widget:






Property	Value
QObject	
objectName	MainWindow
QWidget	
windowModality	NonModal
enabled	<input checked="" type="checkbox"/>
geometry	[(0, 0), 800 x 600]
X	0
Y	0
Width	800
Height	600
sizePolicy	[Preferred, Preferred, 0, 0]
minimumSize	0 x 0
Width	0

Setting up the main window



- Start by resizing the main window and changing the title
 - Resize the main window to 1000x800 – gives us space to work with
 - The title is also nice to change for reference (you can also add an icon, etc..., etc...)

MainWindow : QMainWindow	
Property	Value
▼ QObject	
objectName	MainWindow
▼ QWidget	
windowModality	NonModal
enabled	<input checked="" type="checkbox"/>
▼ geometry	[(0, 0), 1000 x 800]
X	0
Y	0
Width	1000
Height	800
sizePolicy	[Preferred, Preferred, 0, 0]
▼ minimumSize	0 x 0
Width	0

MainWindow : QMainWindow	
Property	Value
mouseTracking	<input type="checkbox"/>
tabletTracking	<input type="checkbox"/>
focusPolicy	NoFocus
contextMenuPolicy	DefaultContextMenu
acceptDrops	<input type="checkbox"/>
▶ windowTitle	MainWindow
▼ windowIcon	
Theme	
Normal Off	
Normal On	
Disabled Off	
Disabled On	

Adding Widgets

➤ We are going to start by adding organization widgets

- ❑ Putting widgets in groups helps with organization and user readability
- ❑ The tab widget creates tabs (like a web browser) for widgets

The screenshot shows the Qt Designer interface with a central canvas titled "POD Filter - main_window.ui*". A red box labeled "Tab Widget" points to a tabbed interface with "Simple Settings" and "Advanced Settings" tabs. A blue box labeled "Group box" points to a "Folder Settings" group box within the "Simple Settings" tab. The left sidebar shows the "Widget Box" with categories like Containers, Input Widgets, and Display Widgets. The right sidebar shows the "Object Inspector" and "Property Editor". The "Object Inspector" shows a hierarchy: MainWindow (QMainWindow) -> centralWidget (QWidget) -> settingsTabWidget (QTabWidget) -> simpleSettingsTab (QWidget) -> folderGroup (QGroupBox). The "Property Editor" shows properties for the selected "tabWidget: QTabWidget", including geometry, sizePolicy, minimumSize, maximumSize, sizeIncrement, baseSize, palette, font, and cursor.

Tab Widget

Group box

Hierarchy shows in this tree (make sure to name things clearly)

Editing Tab Text



➤ Tab titles need to be edited here
(not so obvious)

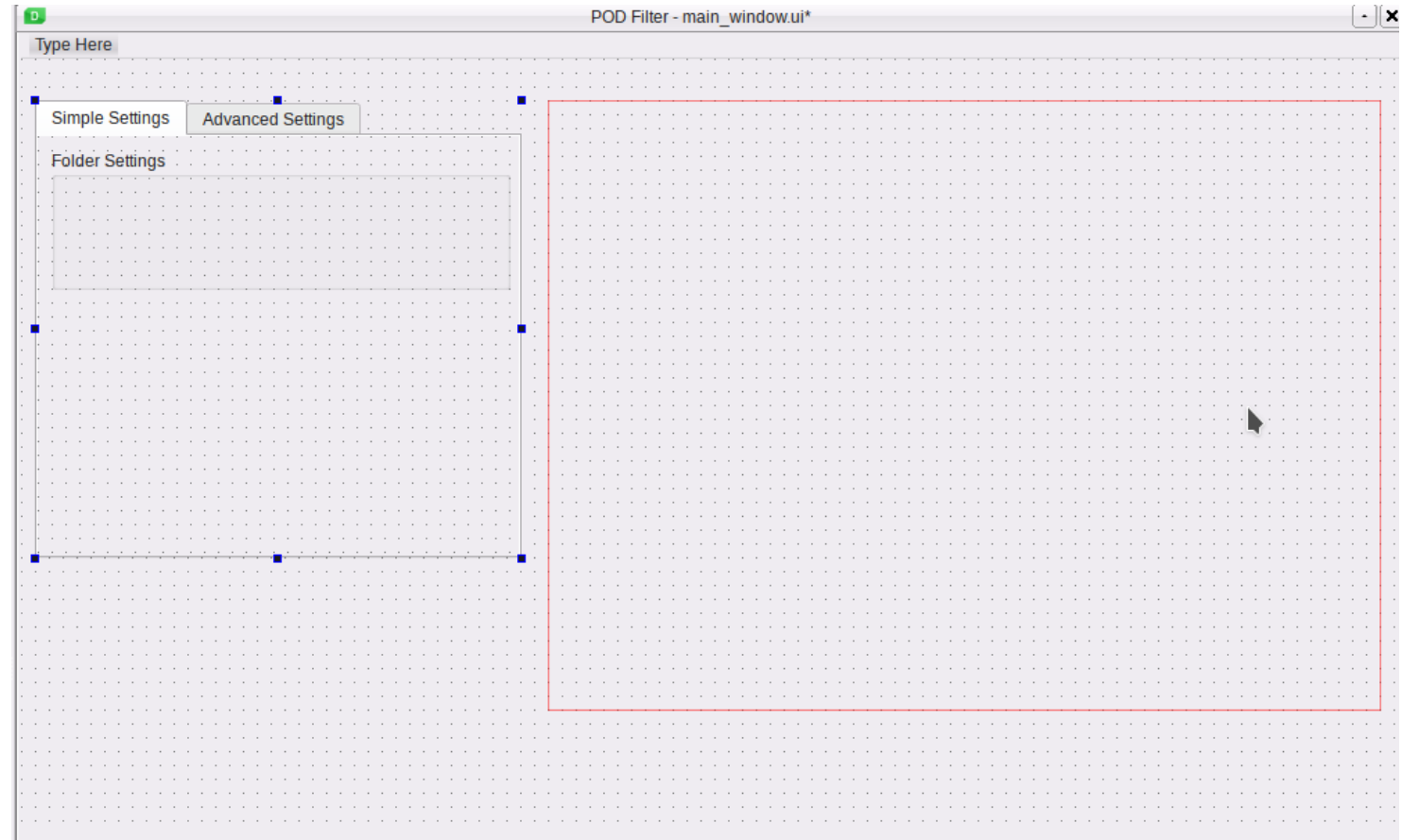
- ☐ Click the tab you want to name
- ☐ Change the "currentTabText" to change the title

tabWidget : QTabWidget	
Property	Value
layoutDirection	LeftToRight
autoFillBackground	<input type="checkbox"/>
styleSheet	
▶ locale	English, United States
▶ inputMethodHints	ImhNone
▼ QTabWidget	
tabPosition	North
tabShape	Rounded
currentIndex	0
▶ iconSize	16 x 16
elideMode	ElideNone
usesScrollButtons	<input checked="" type="checkbox"/>
documentMode	<input type="checkbox"/>
tabsClosable	<input type="checkbox"/>
movable	<input type="checkbox"/>
tabBarAutoHide	
▶ currentTabText	Simple Settings
currentTabName	simpleSettingsTab
▶ currentTabIcon	
▶ currentTabToolTip	
▶ currentTabWhatsThis	

Plot Layout



- This needs to be added for our custom matplotlib widget
- (Red rectangle)



Adding More Widgets



- Let's populate the settings group with some more widgets
- A Label Widget lets the user know what they're looking at
- A text widget is useful for input/output
 - ❑ For these we want to make the box "readOnly"
- The Tool Button will be used to select our folders

The image shows a GUI design tool interface. At the top, there are two tabs: "Simple Settings" and "Advanced Settings". Below them is a "Folder Settings" section with two text input fields labeled "Load Folder" and "Save Folder", each followed by a small button with three dots. In the center, there are three red rounded rectangular buttons labeled "Label", "Text Edit", and "Tool Button". At the bottom right, there is a properties panel with a table of widget properties.

Property	Value
echoMode	Normal
cursorPosition	0
alignment	AlignLeft, AlignVCenter
dragEnabled	<input type="checkbox"/>
readOnly	<input checked="" type="checkbox"/>
placeholderText	
translatable	<input checked="" type="checkbox"/>
disambiguation	
comment	
cursorMoveStyle	LogicalMoveStyle
clearButtonEnabled	<input type="checkbox"/>

Naming Widgets



- It is **very** important to name your widgets properly
 - ❑ Good name: "recordButton,"
"flipImageCheckbox"
 - ❑ Bad name: "petersFancyWidgetThatDoesStuff,"
"widget56"
- When creating scripts, it should be obvious what the widget does
- The name should have
 - ❑ Functionality indicator
 - ❑ What type of widget it is

MainWindow	QMainWindow
centralwidget	QWidget
settingsTabWidget	QTabWidget
simpleSettingsTab	QWidget
folderGroup	QGroupBox
loadFolderButton	QToolButton
loadFolderEdit	QLineEdit
loadFolderLabel	QLabel
saveFolderButton	QToolButton
saveFolderEdit	QLineEdit
saveFolderLabel	QLabel
advancedSettingsTab	QWidget
plotLayout	QVBoxLayout
menubar	QMenuBar
statusbar	QStatusBar

Exporting to a python file



- We have our initial app layout – save and create a .ui file
- To export to a python file run this command
 - "pyuic5 /path/to/ui/file.ui -o /path/to/python/ui/file.py"
 - This generates a file with all of the layouts
- **Do not edit this file**
 - It will get rewritten if you export the layout again

```
main_window.py x main_window_gui.py x app.py x extra_widgets.py x
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'main_window.ui'
4  #
5  # Created by: PyQt5 UI code generator 5.15.7
6  #
7  # WARNING: Any manual changes made to this file will be lost when pyuic5 is
8  # run again. Do not edit this file unless you know what you are doing.
9
10 from PyQt5 import QtCore, QtGui, QtWidgets
11
12 class Ui_MainWindow(object):
13     def setupUi(self, MainWindow):
14         MainWindow.setObjectName("MainWindow")
15         MainWindow.resize(1000, 600)
16         self.centralwidget = QtWidgets.QWidget(MainWindow)
17         self.centralwidget.setObjectName("centralwidget")
18         self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
19         self.tabWidget.setGeometry(QtCore.QRect(10, 30, 351, 331))
20         self.tabWidget.setObjectName("tabWidget")
21         self.simpleSettingsTab = QtWidgets.QWidget()
22         self.simpleSettingsTab.setObjectName("simpleSettingsTab")
23         self.folderGroup = QtWidgets.QGroupBox(self.simpleSettingsTab)
24         self.folderGroup.setGeometry(QtCore.QRect(10, 10, 331, 101))
25         self.folderGroup.setObjectName("folderGroup")
26         self.loadFolderLabel = QtWidgets.QLabel(self.folderGroup)
27         self.loadFolderLabel.setGeometry(QtCore.QRect(10, 30, 91, 21))
28         self.loadFolderLabel.setObjectName("loadFolderLabel")
29         self.loadFolderEdit = QtWidgets.QLineEdit(self.folderGroup)
30         self.loadFolderEdit.setEnabled(False)
31         self.loadFolderEdit.setGeometry(QtCore.QRect(100, 30, 191, 22))
32         self.loadFolderEdit.setObjectName("loadFolderEdit")
33         self.loadFolderButton = QtWidgets.QToolButton(self.folderGroup)
34         self.loadFolderButton.setGeometry(QtCore.QRect(300, 30, 23, 21))
35         self.loadFolderButton.setObjectName("loadFolderButton")
36         self.saveFolderLabel = QtWidgets.QLabel(self.folderGroup)
37         self.saveFolderLabel.setGeometry(QtCore.QRect(10, 60, 91, 21))
38         self.saveFolderLabel.setObjectName("saveFolderLabel")
39         self.saveFolderEdit = QtWidgets.QLineEdit(self.folderGroup)
40         self.saveFolderEdit.setEnabled(False)
41         self.saveFolderEdit.setGeometry(QtCore.QRect(100, 60, 191, 22))
42         self.saveFolderEdit.setObjectName("saveFolderEdit")
43         self.saveFolderButton = QtWidgets.QToolButton(self.folderGroup)
44         self.saveFolderButton.setGeometry(QtCore.QRect(300, 60, 23, 21))
45         self.saveFolderButton.setObjectName("saveFolderButton")
```




Creating our Main Window File

- The exported file creates a class – we want to import this into a separate file
 - If we change the UI file, it will completely rewrite our layout script
 - "Separate the GUI and Business logic"
- Every function will be initialized here
 - A separate class will be created to contain the POD filter functions
 - This helps to keep files small and organized

```
main_window.py X main_window_gui.py X app.py X extra_widgets.py X
6 @author: peter
7 """
8
9 #Import the matplotlib widget and main window
10 from panels.main_window import Ui_MainWindow
11 from gui.extra_widgets import MplWidget
12
13 #Function Classes - we use this class for all of our functions
14 #from functions.functions_pod import PODWrapper
15
16 #Imports for pyqt5 widgets
17 from PyQt5 import QtWidgets, QtCore, QtGui
18 from PyQt5.QtCore import pyqtSlot #This is for threading/signals
19
20 #Dark theme - automatically applies nice stylesheet
21 #import qdarktheme
22
23 #For configuration files
24 from configparser import ConfigParser
25 import numpy as np
26 from pathlib import Path
27
28 import matplotlib
29 matplotlib.use('Qt5Agg')
30
31
32 #Main Window Object
33 class MainWindow(Ui_MainWindow):
34     #Init function (runs on creation)
35     def __init__(self):
36         #Inherit objects from our panel file
37         super(MainWindow, self).__init__()
38
39     #This is important for setting window properties
40     self.window = QtWidgets.QMainWindow()
41
42     #This does all of the layout and attaches widgets to the window object
43     self.setupUi(self.window)
44
45     #Create our plot widget and attach to the plot layout
46     self.livePlotWidget = MplWidget(navigationToolbar = True)
47     self.plotLayout.addWidget(self.livePlotWidget)
48
49     #Set load folder to default
50     self.loadFolder = Path('/')
51
52
```

Import our class here

Create new class and inherit

Embedding a matplotlib plot



- This will be useful later
- The matplotlib widget class (custom made) gets embedded into the GUI
- Enables full functionality of matplotlib (quite useful)

```
main_window.py X main_window_gui.py X app.py X extra_widgets.py X
6  @author: peter
7  """
8
9  #Import the matplotlib widget and main window
10 from panels.main_window import Ui_MainWindow
11 from gui.extra_widgets import MplWidget
12
13 #Function Classes - we use this class for all of our functions
14 #from functions.functions_pod import PODWrapper
15
16 #Imports for pyqt5 widgets
17 from PyQt5 import QtWidgets, QtCore, QtGui
18 from PyQt5.QtCore import pyqtSlot #This is for threading/signals
19
20 #Dark theme - automatically applies nice stylesheet
21 #import qdarktheme
22
23 #For configuration files
24 from configparser import ConfigParser
25 import numpy as np
26 from pathlib import Path
27
28 import matplotlib
29 matplotlib.use('Qt5Agg')
30
31
32 #Main Window Object
33 class MainWindow(Ui_MainWindow):
34     #Init function (runs on creation)
35     def __init__(self):
36         #Inherit objects from our panel file
37         super(MainWindow, self).__init__()
38
39         #This is important for setting window properties
40         self.window = QtWidgets.QMainWindow()
41
42         #This does all of the layout and attaches widgets to the window object
43         self.setupUi(self.window)
44
45         #Create our plot widget and attach to the plot layout
46         self.livePlotWidget = MplWidget(navigationToolbar = True)
47         self.plotLayout.addWidget(self.livePlotWidget)
48
49         #Set load folder to default
50         self.loadFolder = Path('/')
51
52
```

Embed
matplotlib
widget



What app.py does

- app.py is the script that we run – it's good to keep things short and simple
- It has 3 major components
 - Import our custom classes
 - Set the working directory (helps prevent import issues)
 - Create the application instance and tell python to close when the app stops

```
main_window.py X main_window_gui.py X app.py X extra_widgets.py X
1 from pathlib import Path
2 import os, sys
3
4 #Current File Directory - pathlib is great for this
5 FILE_PATH = Path(__file__).parent.absolute()
6 sys.path.insert(0, str(FILE_PATH))
7
8 #Change directory to current file path
9 os.chdir(FILE_PATH)
10
11 from PyQt5 import QtWidgets, QtGui
12 import sys
13
14 #Import our main window file
15 from gui.main_window_gui import MainWindow
16
17
18 def main():
19     #Create application object
20     app = QtWidgets.QApplication(sys.argv)
21
22     #Set application name and version
23     app.setApplicationName('dopplerlidarcontrol')
24     app.setApplicationVersion('0.1')
25
26     #Create main window object
27     mainWindow = MainWindow()
28
29     #Show main window
30     mainWindow.window.show()
31
32     #Stop script on exit
33     sys.exit(app.exec())
34
35
36
37
38 #Run script if it is main
39 if __name__ == '__main__':
40     main()
```

Change directory

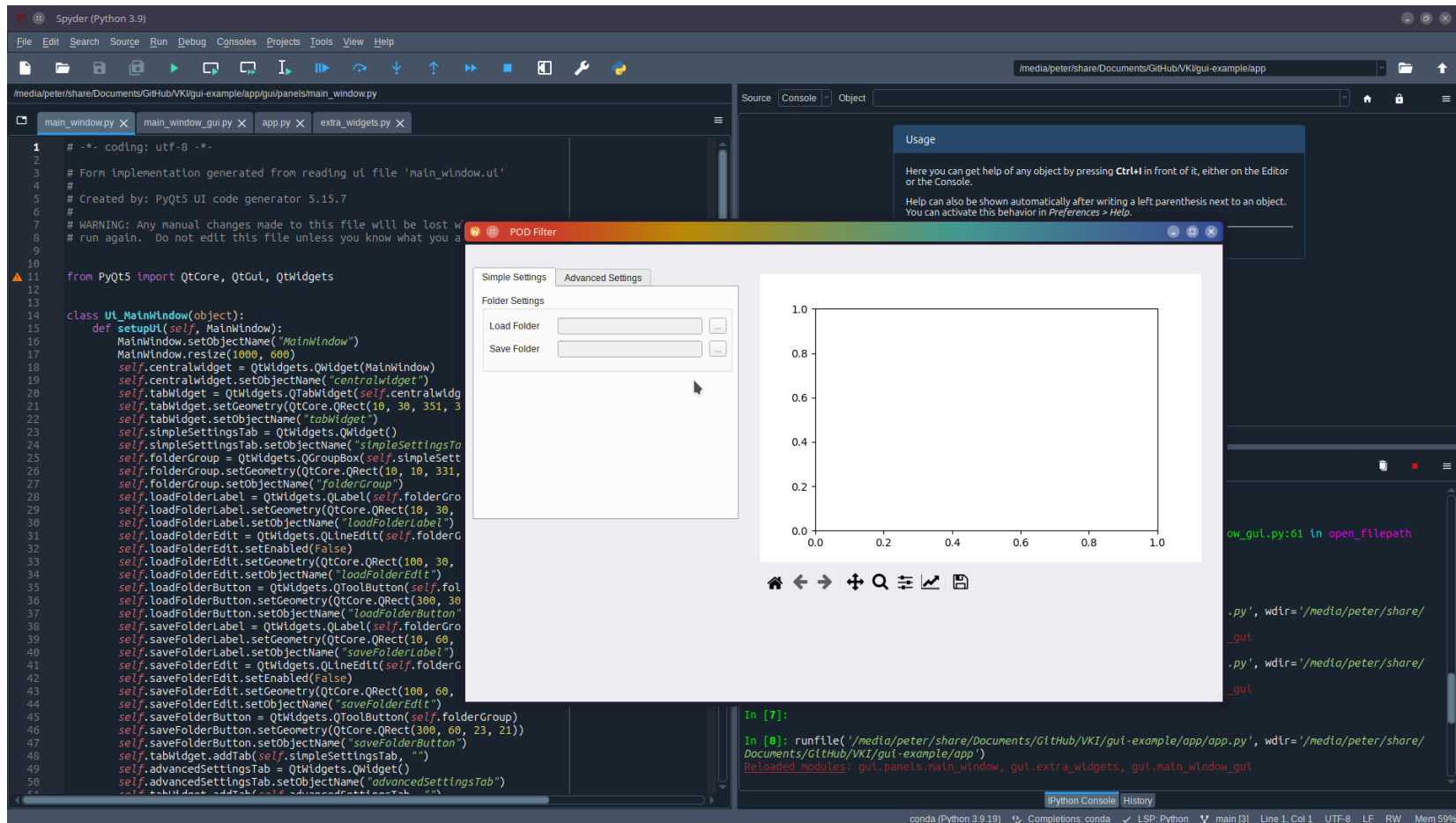
Imports

Create app and close on exit

Running app.py



- Spyder/vs code can run the same app.py
 - When this is run, a new window should appear!





Part 2: Our First Function

Our first function: Selecting an Image Folder



- One of the most useful things for GUIs is to create an easy workflow for commonly used scripts
- Selecting a folder for your workspace is always useful to do
- Pyqt has a built-in function for this

```
#These two functions are basically the same - just for saving/loading...
def open_filepath(self, var):
    #Create a file dialog to open a new folder - return it with a path object - default directory is record folder
    self.loadFolder = Path(QtWidgets.QFileDialog.getExistingDirectory(directory = str(self.loadFolder)))

    #Set the text to the file path
    self.loadFolderEdit.setText(str(self.loadFolder))

def save_filepath(self):
    #Create a file dialog to open a new folder - return it with a path object - default directory is record folder
    self.saveFolder = Path(QtWidgets.QFileDialog.getExistingDirectory(directory = str(self.loadFolder)))

    #Set the text to the file path
    self.saveFolderEdit.setText(str(self.saveFolder))
```



Connecting functions

- The functions need to be connected to the widget that performs the action
- This is quite simple to do!
 - You usually connect the widget action to the function
 - If there's an input variable use the "lambda" functionality
- Run app.py again and the button should open a file dialog

```
#Main Window Object
class MainWindow(Ui_MainWindow):
    #Init function (runs on creation)
    def __init__(self):
        #Inherit objects from our panel file
        super(MainWindow, self).__init__()

        #This is important for setting window properties
        self.window = QtWidgets.QMainWindow()

        #This does all of the layout and attaches widgets to the window
        self.setupUi(self.window)

        #Create our plot widget and attach to the plot layout
        self.livePlotWidget = MplWidget(navigationToolbar = True)
        self.plotLayout.addWidget(self.livePlotWidget)

        #Set load folder to default - this will be changed when we create a config file
        self.loadFolder = Path('/')

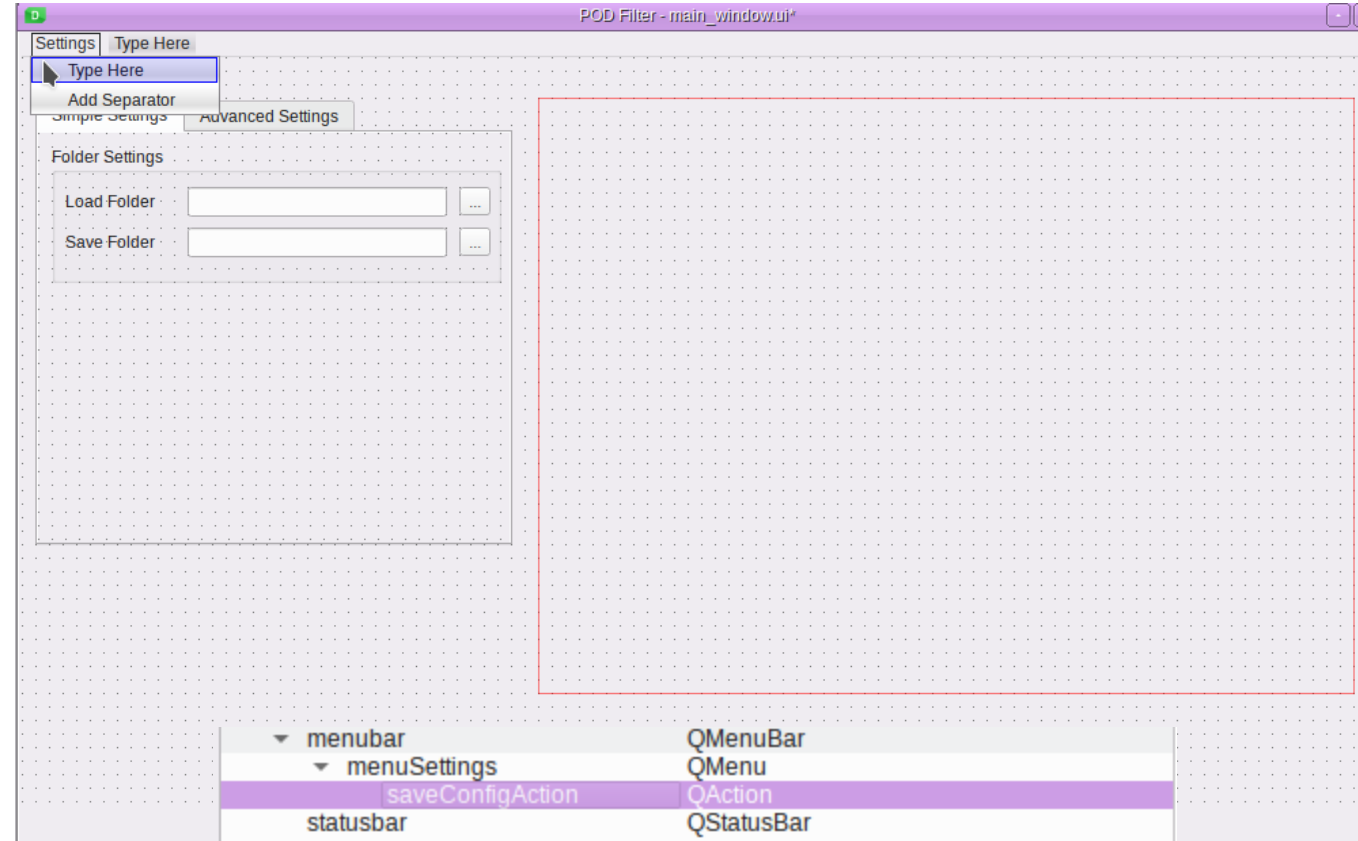
        #Connect buttons to functions
        self.loadFolderButton.clicked.connect(lambda: self.open_filepath(1))
        self.saveFolderButton.clicked.connect(self.save_filepath)
```

lambda:
functionality



Creating a Settings File/Titlebar

- A settings file is useful in the event that your GUI closes/crashes
- Usually this can go on the titlebar of the GUI
- Create a menu entry called Save Config
- We can also add a keyboard shortcut
 - i.e. "ctrl+alt+s"



font	A [Sans Serif, 9]
shortcut	Ctrl+Alt+S
shortcutContext	WindowShortcut

Saving a Settings File



- Now let's create a function for this
- A very useful package "configparser" enables us to create readable configuration files
- Remember to connect the button to the function (or it won't run)

```
class MainWindow(QtWidgets.QMainWindow):
    #Init function (runs on creation)
    def __init__(self):
        #Inherit objects from our panel file
        super(MainWindow, self).__init__()

        #This is important for setting window properties
        self.window = QtWidgets.QMainWindow()

        #This does all of the layout and attaches widgets to the window object
        self.setupUi(self.window)

        #Create our plot widget and attach to the plot layout
        self.livePlotWidget = MplWidget(navigationToolbar = True)
        self.plotLayout.addWidget(self.livePlotWidget)

        #Set load folder to default - this will be changed when we create a config file
        self.loadFolder = Path('/')

        #Connect buttons to functions
        self.loadFolderButton.clicked.connect(lambda: self.open_filepath(1))
        self.saveFolderButton.clicked.connect(self.save_filepath)

        #Connect our status bar actions to functions
        self.saveConfigAction.triggered.connect(self.save_app_config)

        #Automatically load our app configuration
        self.load_app_config()
```

```
#This function saves default settings
def save_app_config(self):
    #Create a config parser object for the settings
    self.settings = configparser.ConfigParser()

    #Folder Settings create a nested dictionary
    self.settings['Folder Settings'] = {}

    #Now lets get the value of the folder boxes
    self.settings['Folder Settings']['LoadFolder'] = self.loadFolderEdit.text()
    self.settings['Folder Settings']['SaveFolder'] = self.saveFolderEdit.text()

    #Save the write the settings to our config file
    with open(CONFIG_PATH, 'w') as settings_file:
        self.settings.write(settings_file)
```

main_window.py × main_window_gui.py* × app.py × extra_widgets.py × app_config.cfg* ×

```
[Folder Settings]
loadfolder = /media/peter/share/Documents/Data/PIV Data/100mus
savefolder =
```



Loading a Settings File

- We can make the GUI automatically load the parameters we save every time we run it
- Also can connect to a button

```
class MainWindow(Ui_MainWindow):
    #Init function (runs on creation)
    def __init__(self):
        #Inherit objects from our panel file
        super(MainWindow, self).__init__()

        #This is important for setting window properties
        self.window = QtWidgets.QMainWindow()

        #This does all of the layout and attaches widgets to the window object
        self.setupUI(self.window)

        #Create our plot widget and attach to the plot layout
        self.livePlotWidget = MplWidget(navigationToolbar = True)
        self.plotLayout.addWidget(self.livePlotWidget)

        #Set load folder to default - this will be changed when we create a config file
        self.loadFolder = Path('/')

        #Connect buttons to functions
        self.loadFolderButton.clicked.connect(lambda: self.open_filepath(1))
        self.saveFolderButton.clicked.connect(self.save_filepath)

        #Connect our status bar actions to functions
        self.saveConfigAction.triggered.connect(self.save_app_config)

        #Automatically load our app configuration
        self.load_app_config()
```

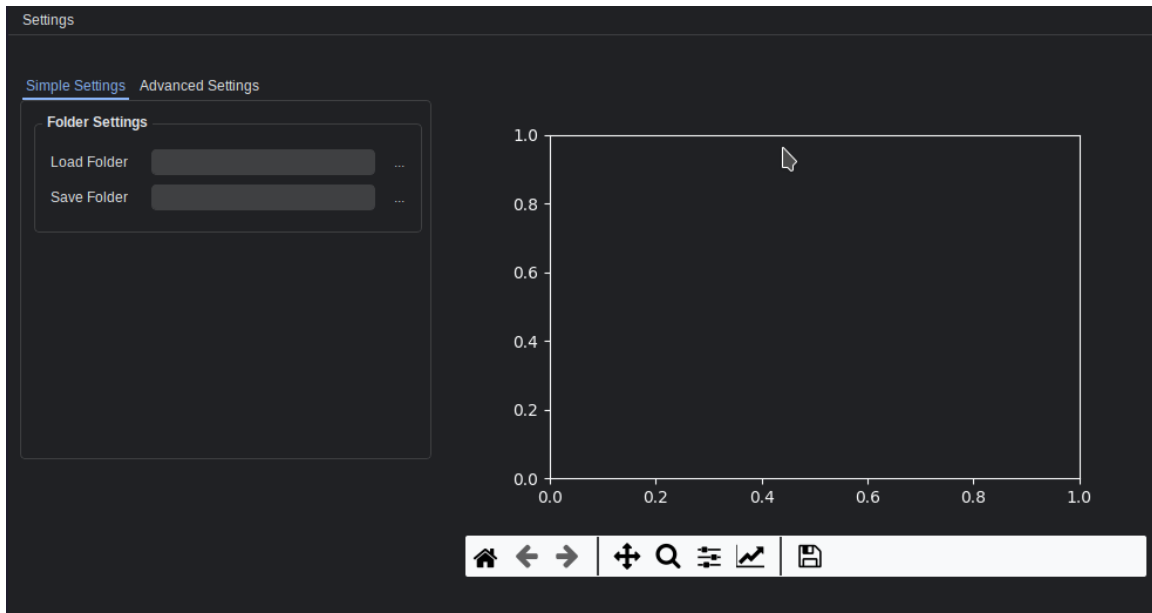
```

}8
}9
}10
}11
}12
}13
}14
}15
}16
}17
}18
}19
}20
}21
}22
}23
}24
}25
}26
}27
}28
}29
}30
}31
}32
}33
}34
}35
}36
}37
}38
}39
}40
}41
}42
}43
}44
}45
}46
}47
}48
}49
}50
}51
}52
}53
}54
}55
}56
}57
}58
}59
}60
}61
}62
}63
}64
}65
}66
}67
}68
}69
}70
}71
}72
}73
}74
}75
}76
}77
}78
}79
}80
}81
}82
}83
}84
}85
}86
}87
}88
}89
}90
}91
}92
}93
}94
}95
}96
}97
}98
}99
}100
}101
}102
}103
}104
}105
}106
}107
}108
}109
}110
}111
}112
}113
}114
}115
}116
}117
}118
}119
}120
}121
}122
}123
}124
}125
}126
}127
}128
}129
}130
}131
}132
}133
}134
}135
}136
}137
}138
}139
}140
}141
}142
}143
}144
}145
}146
}147
}148
}149
}150
}151
}152
}153
}154
}155
}156
}157
}158
}159
}160
}161
}162
}163
}164
}165
}166
}167
}168
}169
}170
}171
}172
}173
}174
}175
}176
}177
}178
}179
}180
}181
}182
}183
}184
}185
}186
}187
}188
}189
}190
}191
}192
}193
}194
}195
}196
}197
}198
}199
}200
}201
}202
}203
}204
}205
}206
}207
}208
}209
}210
}211
}212
}213
}214
}215
}216
}217
}218
}219
}220
}221
}222
}223
}224
}225
}226
}227
}228
}229
}230
}231
}232
}233
}234
}235
}236
}237
}238
}239
}240
}241
}242
}243
}244
}245
}246
}247
}248
}249
}250
}251
}252
}253
}254
}255
}256
}257
}258
}259
}260
}261
}262
}263
}264
}265
}266
}267
}268
}269
}270
}271
}272
}273
}274
}275
}276
}277
}278
}279
}280
}281
}282
}283
}284
}285
}286
}287
}288
}289
}290
}291
}292
}293
}294
}295
}296
}297
}298
}299
}300
}301
}302
}303
}304
}305
}306
}307
}308
}309
}310
}311
}312
}313
}314
}315
}316
}317
}318
}319
}320
}321
}322
}323
}324
}325
}326
}327
}328
}329
}330
}331
}332
}333
}334
}335
}336
}337
}338
}339
}340
}341
}342
}343
}344
}345
}346
}347
}348
}349
}350
}351
}352
}353
}354
}355
}356
}357
}358
}359
}360
}361
}362
}363
}364
}365
}366
}367
}368
}369
}370
}371
}372
}373
}374
}375
}376
}377
}378
}379
}380
}381
}382
}383
}384
}385
}386
}387
}388
}389
}390
}391
}392
}393
}394
}395
}396
}397
}398
}399
}400
}401
}402
}403
}404
}405
}406
}407
}408
}409
}410
}411
}412
}413
}414
}415
}416
}417
}418
}419
}420
}421
}422
}423
}424
}425
}426
}427
}428
}429
}430
}431
}432
}433
}434
}435
}436
}437
}438
}439
}440
}441
}442
}443
}444
}445
}446
}447
}448
}449
}450
}451
}452
}453
}454
}455
}456
}457
}458
}459
}460
}461
}462
}463
}464
}465
}466
}467
}468
}469
}470
}471
}472
}473
}474
}475
}476
}477
}478
}479
}480
}481
}482
}483
}484
}485
}486
}487
}488
}489
}490
}491
}492
}493
}494
}495
}496
}497
}498
}499
}500
}501
}502
}503
}504
}505
}506
}507
}508
}509
}510
}511
}512
}513
}514
}515
}516
}517
}518
}519
}520
}521
}522
}523
}524
}525
}526
}527
}528
}529
}530
}531
}532
}533
}534
}535
}536
}537
}538
}539
}540
}541
}542
}543
}544
}545
}546
}547
}548
}549
}550
}551
}552
}553
}554
}555
}556
}557
}558
}559
}560
}561
}562
}563
}564
}565
}566
}567
}568
}569
}570
}571
}572
}573
}574
}575
}576
}577
}578
}579
}580
}581
}582
}583
}584
}585
}586
}587
}588
}589
}590
}591
}592
}593
}594
}595
}596
}597
}598
}599
}600
}601
}602
}603
}604
}605
}606
}607
}608
}609
}610
}611
}612
}613
}614
}615
}616
}617
}618
}619
}620
}621
}622
}623
}624
}625
}626
}627
}628
}629
}630
}631
}632
}633
}634
}635
}636
}637
}638
}639
}640
}641
}642
}643
}644
}645
}646
}647
}648
}649
}650
}651
}652
}653
}654
}655
}656
}657
}658
}659
}660
}661
}662
}663
}664
}665
}666
}667
}668
}669
}670
}671
}672
}673
}674
}675
}676
}677
}678
}679
}680
}681
}682
}683
}684
}685
}686
}687
}688
}689
}690
}691
}692
}693
}694
}695
}696
}697
}698
}699
}700
}701
}702
}703
}704
}705
}706
}707
}708
}709
}710
}711
}712
}713
}714
}715
}716
}717
}718
}719
}720
}721
}722
}723
}724
}725
}726
}727
}728
}729
}730
}731
}732
}733
}734
}735
}736
}737
}738
}739
}740
}741
}742
}743
}744
}745
}746
}747
}748
}749
}750
}751
}752
}753
}754
}755
}756
}757
}758
}759
}760
}761
}762
}763
}764
}765
}766
}767
}768
}769
}770
}771
}772
}773
}774
}775
}776
}777
}778
}779
}780
}781
}782
}783
}784
}785
}786
}787
}788
}789
}790
}791
}792
}793
}794
}795
}796
}797
}798
}799
}800
}801
}802
}803
}804
}805
}806
}807
}808
}809
}810
}811
}812
}813
}814
}815
}816
}817
}818
}819
}820
}821
}822
}823
}824
}825
}826
}827
}828
}829
}830
}831
}832
}833
}834
}835
}836
}837
}838
}839
}840
}841
}842
}843
}844
}845
}846
}847
}848
}849
}850
}851
}852
}853
}854
}855
}856
}857
}858
}859
}860
}861
}862
}863
}864
}865
}866
}867
}868
}869
}870
}871
}872
}873
}874
}875
}876
}877
}878
}879
}880
}881
}882
}883
}884
}885
}886
}887
}888
}889
}890
}891
}892
}893
}894
}895
}896
}897
}898
}899
}900
}901
}902
}903
}904
}905
}906
}907
}908
}909
}910
}911
}912
}913
}914
}915
}916
}917
}918
}919
}920
}921
}922
}923
}924
}925
}926
}927
}928
}929
}930
}931
}932
}933
}934
}935
}936
}937
}938
}939
}940
}941
}942
}943
}944
}945
}946
}947
}948
}949
}950
}951
}952
}953
}954
}955
}956
}957
}958
}959
}960
}961
}962
}963
}964
}965
}966
}967
}968
}969
}970
}971
}972
}973
}974
}975
}976
}977
}978
}979
}980
}981
}982
}983
}984
}985
}986
}987
}988
}989
}990
}991
}992
}993
}994
}995
}996
}997
}998
}999
}1000
}1001
}1002
}1003
}1004
}1005
}1006
}1007
}1008
}1009
}1010
}1011
}1012
}1013
}1014
}1015
}1016
}1017
}1018
}1019
}1020
}1021
}1022
}1023
}1024
}1025
}1026
}1027
}1028
}1029
}1030
}1031
}1032
}1033
}1034
}1035
}1036
}1037
}1038
}1039
}1040
}1041
}1042
}1043
}1044
}1045
}1046
}1047
}1048
}1049
}1050
}1051
}1052
}1053
}1054
}1055
}1056
}1057
}1058
}1059
}1060
}1061
}1062
}1063
}1064
}1065
}1066
}1067
}1068
}1069
}1070
}1071
}1072
}1073
}1074
}1075
}1076
}1077
}1078
}1079
}1080
}1081
}1082
}1083
}1084
}1085
}1086
}1087
}1088
}1089
}1090
}1091
}1092
}1093
}1094
}1095
}1096
}1097
}1098
}1099
}1100
}1101
}1102
}1103
}1104
}1105
}1106
}1107
}1108
}1109
}1110
}1111
}1112
}1113
}1114
}1115
}1116
}1117
}1118
}1119
}1120
}1121
}1122
}1123
}1124
}1125
}1126
}1127
}1128
}1129
}1130
}1131
}1132
}1133
}1134
}1135
}1136
}1137
}1138
}1139
}1140
}1141
}1142
}1143
}1144
}1145
}1146
}1147
}1148
}1149
}1150
}1151
}1152
}1153
}1154
}1155
}1156
}1157
}1158
}1159
}1160
}1161
}1162
}1163
}1164
}1165
}1166
}1167
}1168
}1169
}1170
}1171
}1172
}1173
}1174
}1175
}1176
}1177
}1178
}1179
}1180
}1181
}1182
}1183
}1184
}1185
}1186
}1187
}1188
}1189
}1190
}1191
}1192
}1193
}1194
}1195
}1196
}1197
}1198
}1199
}1200
}1201
}1202
}1203
}1204
}1205
}1206
}1207
}1208
}1209
}1210
}1211
}1212
}1213
}1214
}1215
}1216
}1217
}1218
}1219
}1220
}1221
}1222
}1223
}1224
}1225
}1226
}1227
}1228
}1229
}1230
}1231
}1232
}1233
}1234
}1235
}1236
}1237
}1238
}1239
}1240
}1241
}1242
}1243
}1244
}1245
}1246
}1247
}1248
}1249
}1250
}1251
}1252
}1253
}1254
}1255
}1256
}1257
}1258
}1259
}1260
}1261
}1262
}1263
}1264
}1265
}1266
}1267
}1268
}1269
}1270
}1271
}1272
}1273
}1274
}1275
}1276
}1277
}1278
}1279
}1280
}1281
}1282
}1283
}1284
}1285
}1286
}1287
}1288
}1289
}1290
}1291
}1292
}1293
}1294
}1295
}1296
}1297
}1298
}1299
}1300
}1301
}1302
}1303
}1304
}1305
}1306
}1307
}1308
}1309
}1310
}1311
}1312
}1313
}1314
}1315
}1316
}1317
}1318
}1319
}1320
}1321
}1322
}1323
}1324
}1325
}1326
}1327
}1328
}1329
}1330
}1331
}1332
}1333
}1334
}1335
}1336
}1337
}1338
}1339
}1340
}1341
}1342
}1343
}1344
}1345
}1346
}1347
}1348
}1349
}1350
}1351
}1352
}1353
}1354
}1355
}1356
}1357
}1358
}1359
}1360
}1361
}1362
}1363
}1364
}1365
}1366
}1367
}1368
}1369
}1370
}1371
}1372
}1373
}1374
}1375
}1376
}1377
}1378
}1379
}1380
}1381
}1382
}1383
}1384
}1385
}1386
}1387
}1388
}1389
}1390
}1391
}1392
}1393
}1394
}1395
}1396
}1397
}1398
}1399
}1400
}1401
}1402
}1403
}1404
}1405
}1406
}1407
}1408
}1409
}1410
}1411
}1412
}1413
}1414
}1415
}1416
}1417
}1418
}1419
}1420
}1421
}1422
}1423
}1424
}1425
}1426
}1427
}1428
}1429
}1430
}1431
}1432
}1433
}1434
}1435
}1436
}1437
}1438
}1439
}1440
}1441
}1442
}1443
}1444
}1445
}1446
}1447
}1448
}1449
}1450
}1451
}1452
}1453
}1454
}1455
}1456
}1457
}1458
}1459
}1460
}1461
}1462
}1463
}1464
}1465
}1466
}1467
}1468
}1469
}1470
}1471
}1472
}1473
}1474
}1475
}1476
}1477
}1478
}1479
}1480
}1481
}1482
}1483
}1484
}1485
}1486
}1487
}1488
}1489
}1490
}1491
}1492
}1493
}1494
}1495
}1496
}1497
}1498
}1499
}1500
}1501
}1502
}1503
}1504
}1505
}1506
}1507
}1508
}1509
}1510
}1511
}1512
}1513
}1514
}1515
}1516
}1517
}1518
}1519
}1520
}1521
}1522
}1523
}1524
}1525
}1526
}1527
}1528
}1529
}1530
}1531
}1532
}1533
}1534
}1535
}1536
}1537
}1538
}1539
}1540
}1541
}1542
}1543
}1544
}1545
}1546
}1547
}1548
}1549
}1550
}1551
}1552
}1553
}1554
}1555
}1556
}1557
}1558
}1559
}1560
}1561
}1562
}1563
}1564
}1565
}1566
}1567
}1568
}1569
}1570
}1571
}1572
}1573
}1574
}1575
}1576
}1577
}1578
}1579
}1580
}1581
}1582
}1583
}1584
}1585
}1586
}1587
}1588
}1589
}1590
}1591
}1592
}1593
}1594
}1595
}1596
}1597
}1598
}1599
}1600
}1601
}1602
}1603
}1604
}1605
}1606
}1607
}1608
}1609
}1610
}1611
}1612
}1613
}1614
}1615
}1616
}1617
}1618
}1619
}1620
}1621
}1622
}1623
}1624
}1625
}1626
}1627
}1628
}1629
}1630
}1631
}1632
}1633
}1634
}1635
}1636
}1637
}1638
}1639
}1640
}1641
}1642
}1643
}1644
}1645
}1646
}1647
}1648
}1649
}1650
}1651
}1652
}1653
}1654
}1655
}1656
}1657
}1658
}1659
}1660
}1661
}1662
}1663
}1664
}1665
}1666
}1667
}1668
}1669
}1670
}1671
}1672
}1673
}1674
}1675
}1676
}1677
}1678
}1679
}1680
}1681
}1682
}1683
}1684
}1685
}1686
}1687
}1688
}1689
}1690
}1691
}1692
}1693
}1694
}1695
}1696
}1697
}1698
}1699
}1700
}1701
}1702
}1703
}1704
}1705
}1706
}1707
}1708
}1709
}1710
}1711
}1712
}1713
}1714
}1715
}1716
}1717
}1718
}1719
}1720
}1721
}1722
}1723
}1724
}1725
}1726
}1727
}1728
}1729
}1730
}1731
}1732
}1733
}1734
}1735
}1736
}1737
}1738
}1739
}1740
}1741
}1742
}1743
}1744
}1745
}1746
}1747
}1748
}1749
}1750
}1751
}1752
}1753
}1754
}1755
}1756
}1757
}1758
}1759
}1760
}1761
}1762
}1763
}1764
}1765
}1766
}1767
}1768
}1769
}1770
}1771
}1772
}1773
}1774
}1775
}1776
}1777
}1778
}1779
}1780
}1781
}1782
}1783
}1784
}1785
}1786
}1787
}1788
}1789
}1790
}1791
}1792
}1793
}1794
}1795
}1796
}1797
}1798
}1799
}1800
}1801
}1802
}1803
}1804
}1805
}1806
}1807
}1808
}1809
}1810
}1811
}1812
}1813
}1814
}1815
}1816
}1817
}1818
}1819
}1820
}1821
}1822
}1823
}1824
}1825
}1826
}1827
}1828
}1829
}1830
}1831
}1832
}1833
}1834
}1835
}1836
}1837
}1838
}1839
}1840
}1841
}1842
}1843
}1844
}1845
}1846
}1847
}1848
}1849
}1850
}1851
}1852
}1853
}1854
}1855
}1856
}1857
}1858
}1859
}1860
}1861
}1862
}1863
}1864
}1865
}1866
}1867
}1868
}1869
}1870
}1871
}1872
}1873
}1874
}1875
}1876
}1877
}1878
}1879
}1880
}1881
}1882
}1883
}1884
}1885
}1886
}1887
}1888
}1889
}1890
}1891
}1892
}1893
}1894
}1895
}1896
}1897
}1898
}1899
}1900
}1901
}1902
}1903
}1904
}1905
}1906
}1907
}1908
}1909
}1910
}1911
}1912
}1913
}1914
}1915
}1916
}1917
}1918
}1919
}1920
}1921
}1922
}1923
}1924
}1925
}1926
}1927
}1928
}1929
}1930
}1931
}1932
}1933
}1934
}1935
}1936
}1937
}1938
}1939
}1940
}1941
}1942
}1943
}1944
}1945
}1946
}1947
}1948
}1949
}1950
}1951
}1952
}1953
}1954
}1955
}1956
}1957
}1958
}1959
}1960
}1961
}1962
}1963
}1964
}1965
}1966
}1967
}1968
}1969
}1970
}1971
}1972
}1973
}1974
}1975
}1976
}1977
}1978
}1979
}1980
}1981
}1982
}1983
}1984
}1985
}1986
}1987
}1988
}1989
}1990
}1991
}1992
}1993
}1994
}1995
}1996
}1997
}1998
}1999
}2000
}2001
}2002
}2003
}2004
}2005
}2006
}2007
}2008
}2009
}2010
}2011
}2012
}2013
}2014
}2015
}2016
}2017
}2018
}2019
}2020
}2021
}2022
}2023
}2024
}2025
}2026
}2027
}2028
}2029
}2030
}2031
}2032
}2033
}2034
}2035
}2036
}2037
}2038
}2039
}2040
}2041
}2042
}2043
}2044
}2045
}2046
}2047
}2048
}2049
}2050
}2051
}2052
}2053
}2054
}2055
}2056
}2057
}2058
}2059
}2060
}2061
}2062
}2063
}2064
}2065
}2066
}2067
}2068
}2069
}2070
}2071
}2072
}2073
}2074
}2075
}2076
}2077
}2078
}2079
}2080
}2081
}2082
}2083
}2084
}2085
}2086
}2087
}2088
}2089
}2090
}2091
}2092
}2093
}2094
}2095
}2096
}2097
}2098
}2099
}2100
}2101
}2102
}2103
}2104
}2105
}2106
}2107
}2108
}2109
}2110
}2111
}2112
}2113
}2114
}2115
}2116
}2117
}2118
}2119
}2120
}2121
}2122
}2123
}2124
}2125
}2126
}2127
}2128
}2129
}2130
}2131
}2132
}2133
}2134
}2135
}2136
}2137
}2138
}2139
}2140
}2141
}2142
}2143
}2144
}2145
}2146
}2147
}2148
}2149
}2150
}2151
}2152
}2153
}2154
}2155
}2156
}2157
}2158
}2159
}2160
}2161
}2162
}2163
}2164
}2165
}2166
}2167
}2168
}2169
}2170
}2171
}2172
}2173
}2174
}2175
}2176
}2177
```

Changing Themes



- This is extra, but there is a useful package that can easily make the GUI look good everywhere
 - Besides, who doesn't like dark theme
- This goes nicely in the titlebar
- Automatically load the theme by adding it to the config



```
#Quick theme changer - makes everything look good at once
def change_theme(self, theme):
    qdarktheme.setup_theme(theme)

#Change theme for plots
self.livePlotWidget.change_theme(theme)
```

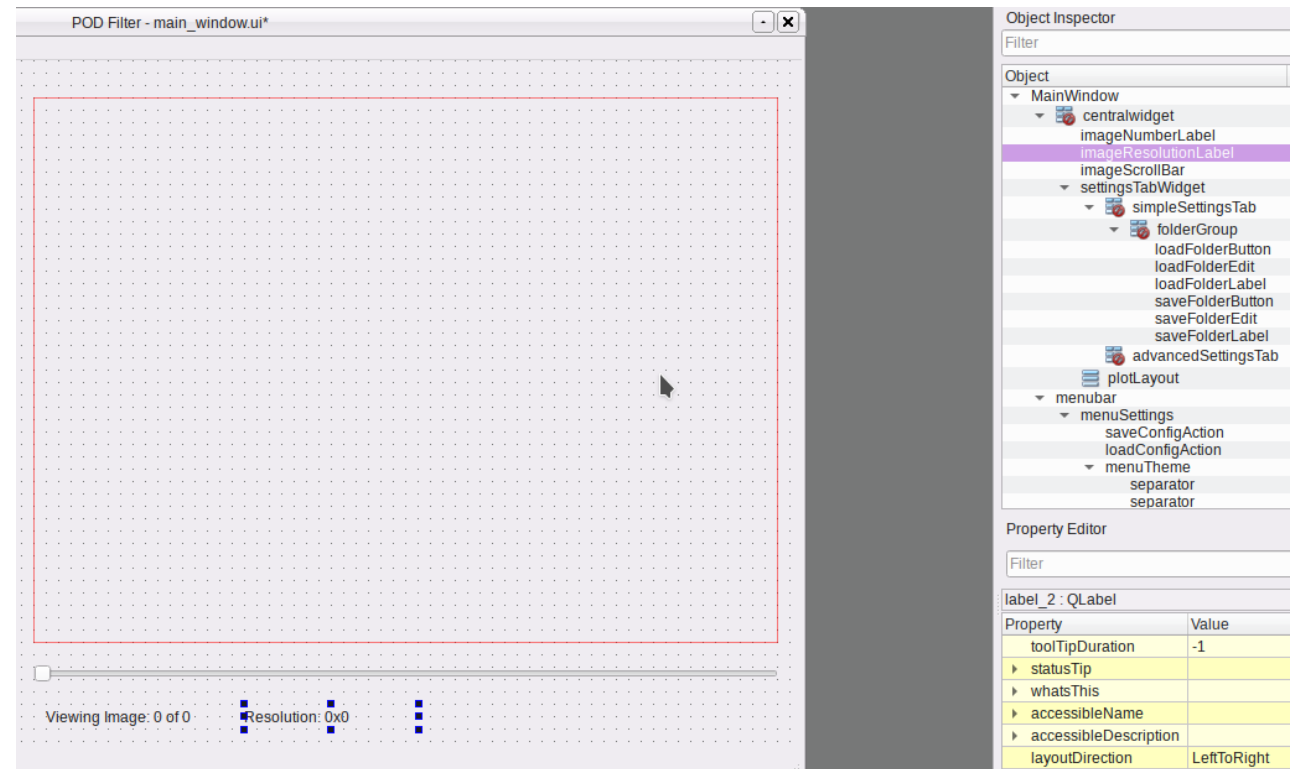
```
self.actionLight.triggered.connect(lambda: self.change_theme('light'))
self.actionDark.triggered.connect(lambda: self.change_theme('dark'))
```

Add these buttons and connect

Looking at Images



- Now let's set up our matplotlib widget!
- Our goal is to inspect the images **before** and **after** the POD
 - This is a big undertaking, but we can start small
 - Let's first create a tool to view the images in our folder
 - This function can be run **without threading** because loading an image won't take long
- Let's add a slider and a couple labels below our plot
- Set all of them to "disabled"



Looking at Images



➤ Now let's create our functions

- First we need to figure out if there are image files present in the folder
- We create a sorted list of them

➤ Then we can update the widgets accordingly

- Set the maximum scroll bar value to the image list length

```
#This tells the app the image files in our folder
def get_images(self):
    #List all files in the image folder and sort
    self.imageList = sorted([file for file in self.loadFolder.glob('*.tif')])

    #Update widgets if there are images
    if len(self.imageList) > 0:
        #Set the scroll bar maximum
        self.imageScrollBar.setMaximum(len(self.imageList))

        #Enable the widgets
        self.imageScrollBar.setEnabled(True)
        self.imageNumberLabel.setEnabled(True)

        #Set the label text
        self.imageNumberLabel.setText('Image Number: %i of %i'%(0, len(self.imageList)))

        #Update with the initial image
        self.update_image(0)

    #Disable the widgets if there arent any images
    else:
        #This can be simplified if they're all in the same group
        self.imageScrollBar.setDisabled(True)
        self.imageNumberLabel.setDisabled(True)
        self.imageResolutionLabel.setDisabled(True)
```

Looking at Images



➤ Now let's create our functions

- Next we create the function to update our plot

➤ This function doesn't take much time to run

- We try to read the image and then plot using our "canvas" (usual functions for matplotlib apply)
- Then we draw the figure
- And update the labels

➤ Don't forget to connect widgets!

- (Done correctly in finished file)

```
# When the scrollbar is changed, the plot will show a new image
def update_image(self, num):
    # Read the image
    try:
        image = imread(self.imageList[num], as_gray=True)

    except:
        return

    # Clear axis - usually not the fastest option
    self.livePlotWidget.cla()

    # Show the image
    self.livePlotWidget.canvas.ax.imshow(image, cmap = 'grey')

    # Turn off the axes
    self.livePlotWidget.canvas.ax.axis('off')

    self.livePlotWidget.canvas.figure.tight_layout()

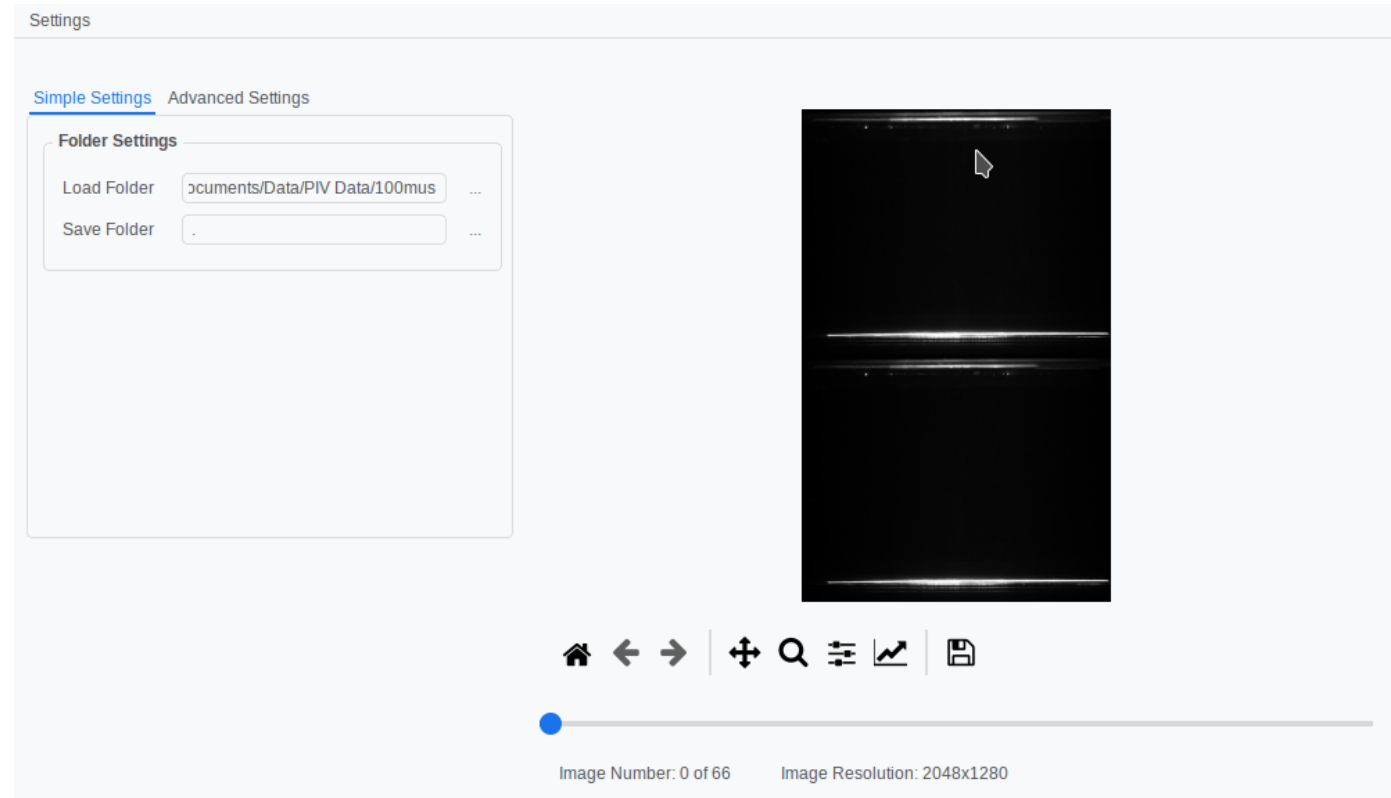
    # Redraw the figure
    self.livePlotWidget.canvas.draw()

    # Update image number on the label
    self.imageNumberLabel.setText('Image Number: %i of %i'%(num, len(self.imageList)))
```

Looking at Images



- When we open our image folder, the images should appear!
- We should be able to scroll and see new image pairs





Part 3: A Long Running Function

Cutting Image Pairs

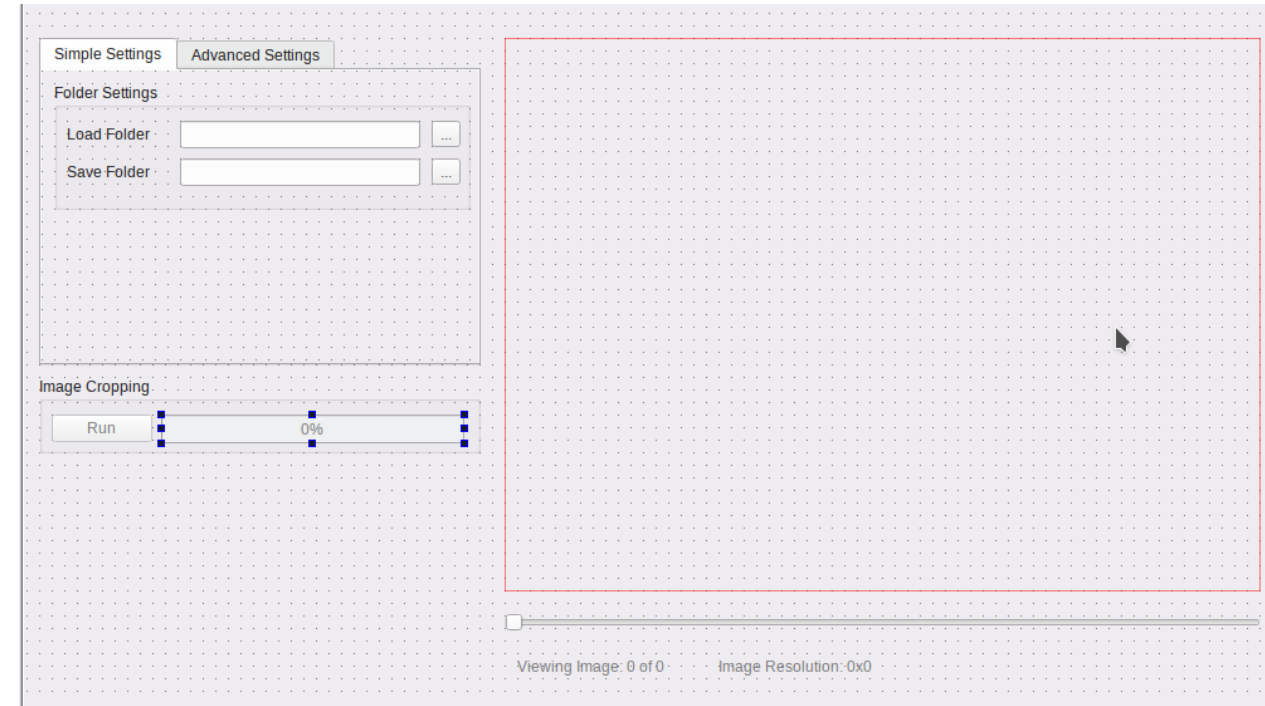


➤ Let's cut our images

- Saving a bunch of image pairs takes time...
- This requires us to use threading/multiprocessing so we can still interact with our GUI

➤ As before... let's create our widgets

- This time we need a button and a progress bar
- Set **only the group box** to disabled (we enable them when we load images)





Cutting Image Pairs

- This script will require threading...
 - pyqt has a built in functionality for this
 - We will also illustrate the signal/slot mechanism to update widgets
- Lets make a new file that we will import into the GUI
 - (POD_functions.py)
 - We need to import the QThread object and the pyqtSignal object

```
main_window.py × main_window_gui.py × app.py × extra_widgets.py × pod_functions.py ×  
  
#This is for threading and connecting widgets  
from PyQt5.QtCore import QThread, pyqtSignal  
  
#This is for reading images  
from skimage.io import imread, imshow, imsave # this is for Matlab users  
  
from pathlib import Path  
import numpy as np
```



Cutting Image Pairs

- We then define a new class
 - It inherits the QThread object to enable threading
- We need to define some signal objects (these connect the GUI output to actions in the thread)
 - They can take in any number/type of object
 - We will connect these to functions in the main GUI

```
class ImageCutter(QThread):  
    #This is our signal that takes a number  
    updateSignal = pyqtSignal(float)  
  
    #This tells us that the thread is finished and if it succeeded  
    finished = pyqtSignal(bool)  
  
    def __init__(self, imageList, saveFolder):  
        #Inherit the QThread Class  
        super(ImageCutter, self).__init__()  
  
        #We need to know our save folder and the list of our image paths  
        self.imageList = imageList  
        self.saveFolder = saveFolder / 'cut'  
  
        #Make the folder if it doesn't exist  
        self.saveFolder.mkdir(exist_ok = True)
```

Cutting Image Pairs



- We can create our image cutting function
 - Takes some time to cut and save all of the images
- In our loop we update the progress bar with our current progress
- Afterwards, we emit our finished signal if the task finished successfully or not
- The "run" function is what gets run when we call `thread.start()`

```
#This function does the cropping
def cut_images(self):
    #This does the cropping
    try:
        for ii, image in enumerate(self.imageList):
            #Read the image and get the shape
            imageArray = imread(image)
            imageShape = imageArray.shape[0]

            #Create image names for each pair
            imageNameA = self.saveFolder / ('cut_%04d_a.tif'%ii)
            imageNameB = self.saveFolder / ('cut_%04d_b.tif'%ii)

            #Save the image pairs
            imsave(imageNameA, imageArray[0:imageShape//2, :])
            imsave(imageNameB, imageArray[imageShape//2:imageShape, :])

            #Update the progress bar
            self.updateSignal.emit(ii)

        #Makes the progress bar satisfying
        self.updateSignal.emit(len(self.imageList))
        self.finished.emit(True)

    except:
        self.finished.emit(False)
        return

#This is what runs in the thread
def run(self):
    self.cut_images()
```



Cutting Image Pairs

- Now in main_window_gui.py
- We create 3 new functions
- cut_images
 - Creates an instance of our new cutter class
 - We connect our signals to the other functions
 - Then we start the thread
- Other functions
 - update_cut_bar updates the progress bar during the loop
 - update_cut_folder will automatically re-load the image folder if the task is successful
- **Don't forget to connect the run button to the cut_images function!**

```
def cut_images(self):
    #Create image cropper class
    self.imageCutter = ImageCutter(self.imageList, self.saveFolder)

    #Connect the signal to updating progress bar
    self.imageCutter.updateSignal.connect(self.update_cut_bar)
    self.imageCutter.finished.connect(self.update_cut_folder)

    #Start thread
    self.imageCutter.start()

def update_cut_bar(self, ii):
    #Set the percentage
    self.cuttingProgressBar.setValue(ii/len(self.imageList)*100)

    #Update the progress
    self.cuttingProgressBar.setFormat('Image %i of %i'%(ii, len(self.imageList)))

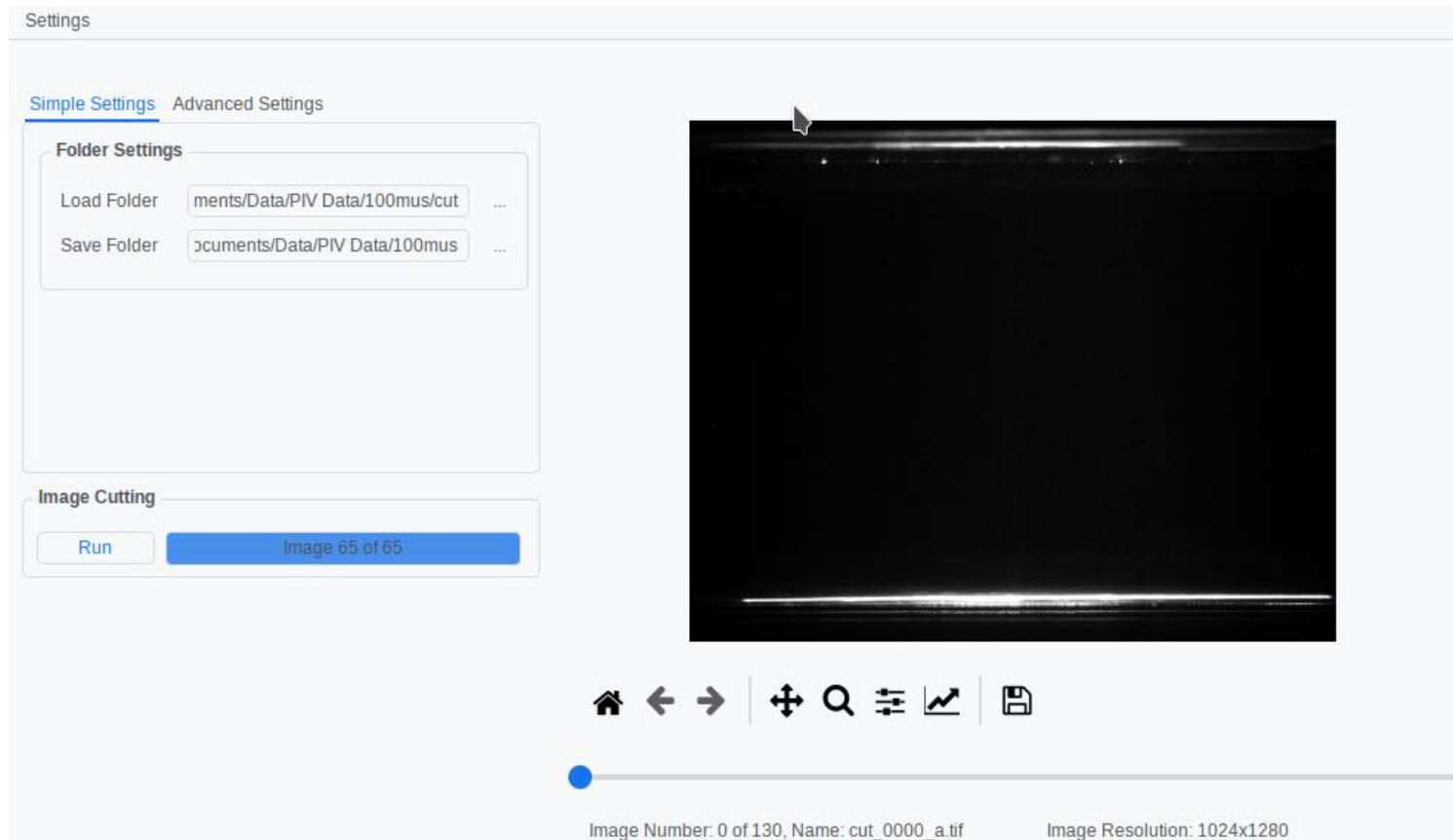
def update_cut_folder(self, finished):
    #Update the folder and re-load the images if the crop works
    if finished:
        self.loadFolder = self.imageCutter.saveFolder
        self.loadFolderEdit.setText(str(self.loadFolder))

        self.get_images()
```

Cutting Image Pairs



- Now we try this!
- The progress bar should update as our task runs!





Part 4: Implement POD

POD



- Now for the big function!
 - We will use everything we have been doing before to do this
- Of course... let's make our widgets first
 - We have a lot of settings to work through
 - We use the spin box to enter integers
 - "Double spin box exists for doubles..."

POD Settings

Number of Modes: 0 ☐ Flip Image

Number of Pairs: 0

Crop:

X: 0 0

Y: 0 0

Image Cutting

Run Cutting Progress

POD Filter

Compute Computation Progress

Save Saving Progress

POD Thread



➤ We create our thread object (like before)

- There are 2 different functions - we switch between by changing a variable in the class
- This also splits the images by a/b if named correctly

```
class PODRunner(QThread):
    #Our signals to update progress bars
    updateSignal = pyqtSignal(float, str)
    saveSignal = pyqtSignal(float, str)

    #Our signal to indicate if things finished
    finishedComputation = pyqtSignal(bool)
    finishedSaving = pyqtSignal(bool)

    def __init__(self, imageList, saveFolder, settings):
        super(PODRunner, self).__init__()

        #We need to know our save folder and the list of our image paths
        self.imageList = imageList
        self.saveFolder = saveFolder / 'POD_Output'

        #Make the folder if it doesn't exist
        self.saveFolder.mkdir(exist_ok = True)

        #Split the list into a and b
        self.imageAList = []
        self.imageBList = []

        for image in self.imageList:
            if image.name[-5] == 'a':
                self.imageAList.append(image)
            elif image.name[-5] == 'b':
                self.imageBList.append(image)

        #Grab information from settings dictionary
        # Number of modes to remove. If 0, the filter is not active!
        self.nModes = settings['nModes']
        self.nPairs = settings['nPairs']
        self.flipImage = settings['flipImage']

        #Crop list - [X1, X2, Y1, Y2]
        self.cropList = settings['cropList']

        #This variable changes if we want to save/compute the filtered matrix
        self.function = 'compute_matrix'
```

POD Thread (cont)



- Our big function does all of the POD (see machine learning/DDMA lectures)
- We emit a string and percentage for the progress bar
- The projection matrices are the only ones that are stored

```
def compute_filtered_matrix(self):
    #Prepare image matrix and calculate shape
    imInitial = imread(self.imageAList[0])

    #Create image crop
    croppedImage = imInitial[self.cropList[2]:self.cropList[3], self.cropList[0]:self.cropList[1]]

    #Flip the image if we want to
    if self.flipImage:
        croppedImage = np.fliplr(croppedImage)

    #Get the shape of the image
    self.imageShape = croppedImage.shape
    ny, nx = self.imageShape

    #Create matrix to concatenate images
    D_a = np.zeros((nx * ny, self.nPairs)) # Initialize the Data matrix for image sequences A.
    D_b = np.zeros((nx * ny, self.nPairs)) # Initialize the Data matrix for image sequences B.

    #Update progress
    self.updateSignal.emit(0, 'Start Processing')

    #Process image pairs and concatenate
    for k in range(0, self.nPairs):
        #Read images
        imA = imread(self.imageAList[k])
        imB = imread(self.imageBList[k])

        #Create image crop
        cropA = imA[self.cropList[2]:self.cropList[3], self.cropList[0]:self.cropList[1]]
        cropB = imB[self.cropList[2]:self.cropList[3], self.cropList[0]:self.cropList[1]]

        #Flip the image if we want to
        if self.flipImage:
            cropA = np.fliplr(cropA)
            cropB = np.fliplr(cropB)

        #Cast to float array
        cropA = np.float64(cropA) # We work with floating number not integers
        cropB = np.float64(cropB) # We work with floating number not integers

        # Reshape into a column Vector
        cropA = np.reshape(cropA, ((nx * ny, 1)))
        cropB = np.reshape(cropB, ((nx * ny, 1)))

        #Append to matrix
        D_a[:, k] = cropA[:, 0]
        D_b[:, k] = cropB[:, 0]
```

POD Thread (cont)



- The saving images thread is very simple
- It reshapes the image from the stored projection matrix and saves it

```
def save_images(self):
    (ny, nx) = self.imageShape
    for k in range(0, self.nPairs):
        #Save A images
        imdA = np.copy(self.D_a_filt[:, k])
        imPODA = np.reshape(imdA, ((ny, nx)))
        imPODA[imPODA < 0] = 0 # Things below 0 are treated as zero

        imPODA = np.uint8(imPODA)
        imsave(self.saveFolder / ('POD_Filt_%03d_a.tif'%k), imPODA)

        #Save b images
        imdB = np.copy(self.D_b_filt[:, k])
        imPODB = np.reshape(imdB, ((ny, nx)))
        imPODB[imPODB < 0] = 0 # Things below 0 are treated as zero

        imPODB = np.uint8(imPODB)
        imsave(self.saveFolder / ('POD_Filt_%03d_b.tif'%k), imPODB)

        #Update the signal
        self.saveSignal.emit(k/self.nPairs*100, 'Saving Image Pair %i of %i'%(k, self.nPairs))

    self.saveSignal.emit(100, 'Finished Saving')
    self.finishedSaving.emit(True)
```

POD In the GUI



- After importing the thread object we need to create functions to run the functions for saving/computing
 - In the function we pass the parameters from the boxes to a podRunner object
 - Then the signals are connected to the relevant functions
 - Then we set the function to "compute_matrix"
 - Then we start the thread
- The user can only save images once the POD is computed
 - This is done by enabling/disabling widgets

```
def compute_pod_matrices(self):
    #Collect settings
    settings = {}
    settings['nModes'] = self.podModeBox.value()
    settings['nPairs'] = self.podPairBox.value()
    settings['flipImage'] = self.podFlipImageCheckbox.isChecked()

    #Crop list - [X1, X2, Y1, Y2]
    settings['cropList'] = [self.xCropMinBox.value(),
                           self.xCropMaxBox.value(),
                           self.yCropMinBox.value(),
                           self.yCropMaxBox.value()]

    #Create podRunner object
    self.podRunner = PODRunner(self.imageList, self.saveFolder, settings)

    #Connect signals to functions
    self.podRunner.updateSignal.connect(self.update_pod_bar)
    self.podRunner.saveSignal.connect(self.update_pod_save_bar)
    self.podRunner.finishedComputation.connect(self.on_finished_computing)
    self.podRunner.finishedSaving.connect(self.on_finished_saving)

    #Set the function to compute matrix and start
    self.podRunner.function = 'compute_matrix'
    self.podRunner.start()

def save_pod_images(self):
    #This sets the podRunner to save images and starts the thread
    self.podRunner.function = 'save'
    self.podRunner.start()
```

POD in the GUI (cont)



- The progress bars also need to be updated
- The gui also needs to do something if the saving/computing is successful

```
def on_finished_computing(self, finished):
    if finished:
        self.podSaveButton.setEnabled(True)
        self.podSaveProgress.setEnabled(True)
        self.showComputedImagesCheckbox.setEnabled(True)

def on_finished_saving(self, finished):
    if finished:
        self.loadFolder = self.podRunner.saveFolder
        self.loadFolderEdit.setText(str(self.loadFolder))

        self.get_images()

#Functions to update POD progress bars
def update_pod_bar(self, percent, label):
    #Set the percentage
    self.podProgressBar.setValue(percent)

    #Update the progress
    self.podProgressBar.setFormat(label)

def update_pod_save_bar(self, percent, label):
    #Set the percentage
    self.podSaveProgress.setValue(percent)

    #Update the progress
    self.podSaveProgress.setFormat(label)
```

POD in the GUI (cont)



- We also can edit the `update_image` function to preview
 - **Bug:** this does not show all of the images, likely to do with `num > len(D_a_filt)`

```
#Preview images from POD if we want to
if self.showComputedImagesCheckbox.isChecked():
    #Select image pair
    if num%2==1: #even images are b
        image = np.copy(self.podRunner.D_b_filt[:, num])
    else:
        image = np.copy(self.podRunner.D_a_filt[:, num])

    (nx, ny) = self.podRunner.imageShape
    image = np.reshape(image, ((ny, nx)))
    image[image < 0] = 0 # Things below 0 are treated as zero

    image = np.uint8(image)
```


POD in the GUI (cont)



- We need to enable the right widgets when we
- Also need to connect widgets
- Overall, this gets tedious more than complicated

```
#This tells the app the image files in our folder
def get_images(self):
    #List all files in the image folder and sort
    self.imageList = sorted([file for file in self.loadFolder.glob(IMAGE_TYPE)])

    #Update widgets if there are images
    if len(self.imageList) > 0:
        #Set the scroll bar maximum
        self.imageScrollBar.setMaximum(len(self.imageList)-1)

        #Enable the widgets
        self.imageScrollBar.setEnabled(True)
        self.imageNumberLabel.setEnabled(True)
        self.imageResolutionLabel.setEnabled(True)

        #Image crop group
        self.cuttingGroup.setEnabled(True)
        self.cuttingProgressBar.setEnabled(True)
        self.cuttingRunButton.setEnabled(True)

        #POD Group
        self.podFilterGroup.setEnabled(True)
        self.podRunButton.setEnabled(True)
        self.podProgressBar.setEnabled(True)
        self.podSettingsBox.setEnabled(True)

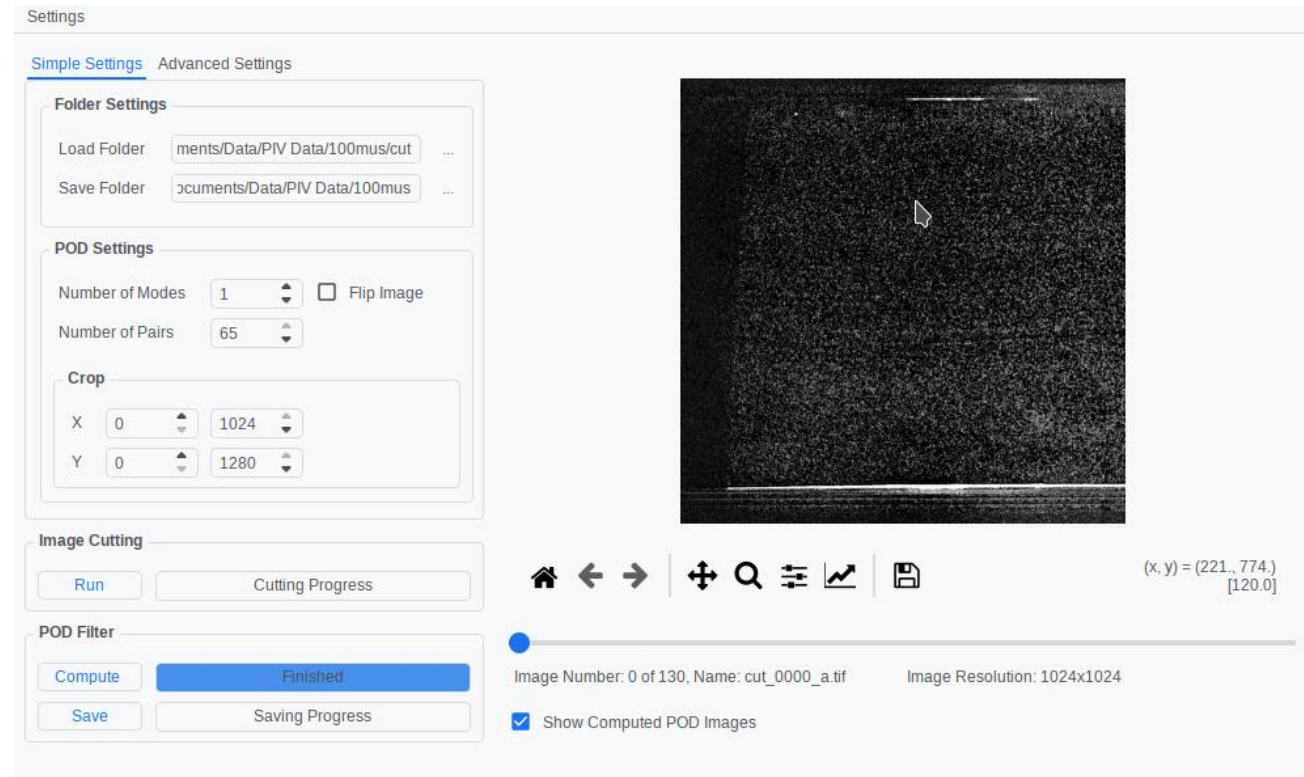
    #Set maximum number of pairs
```

```
#POD Buttons
self.podRunButton.clicked.connect(self.compute_pod_matrices)
self.podSaveButton.clicked.connect(self.save_pod_images)
self.showComputedImagesCheckbox.clicked.connect(self.update_image)
```

POD Finished (mostly)



- This is what it looks like when it is done!
- There are some bugs/features that are left as an exercise to the reader
 - **Feature:** implement real-time feedback on cropping images
 - **Feature:** implement naming folders/a "workspace"
 - **Bug:** previewing POD images does not show both "a" and "b" images
- You can preview the POD computation before saving and save the images





Extra Slides

Appendix: Installing Requirements



- Create a new conda environment (conda create -n envname python=3.9)
 - Use python3.9 for compatibility
- Install pip "conda install pip"
- The main folder has a requirements.txt file
 - Install using "pip install -r requirements.txt"

Python Frameworks



- There are many frameworks available to create GUIs
 - ❑ PyQt and tkinter are the biggest
- PyQt is preferred
 - ❑ Has a nice widget editor
 - ❑ Better multiprocessing support
 - ❑ Compiled in c++
- PySide vs PyQt
 - ❑ Both use Qt

